Highly Dependable Systems

Sistemas de Elevada Confiabilidade

2022-2023

HDS Ledger
Stage 2

Goals

The goal of the second stage of the project is to develop an application (i.e. a simplified Smart Contract) that implements a Token Exchange System (TES). Overall, the application should leverage the existing functionality, developed in the first stage of the project, to build a State Machine Replication system that supports the TES.

Design Requirements

The Token Exchange System holds a set of accounts, where each account has the following properties:

- It is uniquely identified by a public key from a <public,private> key pair
- It holds the current balance

The servers are responsible for maintaining the set of TES accounts and ensure a set of dependability properties discussed below.

A client of the system can perform transfers between a pair of accounts provided it owns the corresponding private key of the crediting account (the one from which money is withdrawn). The set of all accounts and the associated balance should satisfy the following dependability and security guarantees:

- the balance of each account should be non-negative
- the state of the accounts cannot be modified by unauthorized users
- the system should guarantee the non-repudiation of all operations issued on an account

The client API has the following specification:

- create_account(PublicKey key,…)
  Specification: create a new account in the system with the associated public *key,* before first use. In particular, it should make the necessary initializations to enable the first use of the TES system. The account should start with a pre-defined positive balance.
- transfer(PublicKey source, PublicKey destination, int amount, …)

Specification: send a given amount from account *source* to account *destination*, if the balance of the source allows it. If the server responds positively to this invocation, it must be guaranteed that the *source* has the authority to perform the transfer.

- check_balance(PublicKey key,…)
  Specification: obtain the balance of the account associated with the *key* passed as input.

In this stage, we retain the simplifying assumption that the faulty leader protocol and code paths (i.e., round changes) do not need to be implemented, but other than this assumption, every node in the system (servers and clients) might be Byzantine.

Students must reason about the potential attacks that a Byzantine server or client can make and discuss and implement the appropriate countermeasures.

There is a high demand for transactions in the blockchain and performing consensus for a single transaction is expensive. Therefore, for performance reasons, transactions must be grouped in a block that aggregates a set of transactions. For simplicity students can define a fixed block size contained a given set of transactions, for example 10 transactions per block.

Moreover, the operations that only query (and do not update) the state of the system should be able to run a simplified protocol that supports both the following consistency modes:

- Strongly consistent read: allow reads to execute faster than normal blockchain operations while still obeying the same atomic (linearizable) semantics that are provided by the Istanbul BFT protocol.
- Weakly consistent read: allow for reads to provide a correct but stale output, in a way that such reads can still execute in weak connectivity scenarios, namely when the client can only connect to a single replica.

When performing these operations, the client should explicitly choose the mode it desires. Aside from correctness, these read operations should also be concerned with performance. In particular, a naïve solution where the client is sent the whole blockchain to locally obtain the weakly consistent ready reply is not allowed.

Finally, all update transactions must pay a fee to the block producer (the fixed leader given our simplifying assumptions). Students must reason about the implications of these requirements and design the system accordingly.

Implementation Steps

To help in the design and implementation task, we suggest that students break up the project into a series of steps, and thoroughly test each step before moving to the next one. Having an automated build and testing process (e.g.: JUnit) will help students progress faster. Here is a suggested sequence of steps:

- Step 1: Implement the maintenance of the TES state on the server side. More precisely, the TES holds a set of accounts that should be updated whenever a new block is produced.
- Step 2: Define the syntax and semantics of operations, implement the operations on the client side, and test the client and server interactions.
- Step 3: Implement the optimized read-only operations.
- Step 4: Implement a simple client to to showcase the functionality of the system and a set of tests to demonstrate the system robustness to Byzantine attacks.

Submission

Submission will be done through Fénix. The submission shall include:

- a self-contained zip archive containing the source code of the project and any additional libraries required for its compilation and execution. The archive shall also include a set of demo applications/tests that demonstrate the mechanisms integrated in the project to tackle security and dependability threats (e.g., detection of Byzantine behavior from blockchain members). A README file explaining how to run the demos/tests is mandatory.
- a concise report of up to 8,000 characters addressing:
  - explanation and justification of the design, including an explicit analysis of the possible threats and corresponding protection mechanisms.
  - explanation of the dependability guarantees provided by the system.

The deadline is **April 14 at 23:59**. More instructions on the submission will be posted in the course page.

Ethics

The work is intended to be done exclusively by the students in the group and the submitted work should be new and original. It is fine, and encouraged, to discuss ideas with colleagues – that is how good ideas and science happens - but looking and/or including code or report material from external sources (other groups, past editions of this course, other similar courses, code available on the Internet, ChatGPT, etc) is forbidden and considered fraud. We will strictly follow IST policies and any fraud suspicion will be promptly reported.